Computing the transitive closure of a union of affine integer tuple relations

Anna Beletska, Denis Barthou, Wlodzirmierz Bielecki, Albert Cohen

Computing the transitive closure of a union of affine integer tuple relations

Anna Beletska[1], Denis Barthou[2], Wlodzimierz Bielecki[3], and Albert Cohen[4]

[1] INRIA Saclay, France, Anna.Beletska@inria.fr
[2] U. of Versailles St. Quentin, France, Denis.Barthou@prism.uvsq.fr
[3] Technical University of Szczecin, Poland, WBielecki@wi.ps.pl
[4] INRIA Saclay, France, Albert.Cohen@inria.fr

Abstract. *This paper proposes a method to compute the transitive closure of a union of affine relations on integer tuples. Within Presburger arithmetics, complete algorithms to compute the transitive closure exist for convex polyhedra only. In presence of non-convex relations, there exist little but special cases and incomplete heuristics. We introduce a novel sufficient and necessary condition defining a class of relations for which an exact computation is possible. Our method is immediately applicable to a wide area of symbolic computation problems. It is illustrated on representative examples and compared with state-of-the-art approaches.*

1 Introduction

Computing the transitive closure of graphs is an operation underlying many algorithms, from CAD, software engineering, real-time process control, data bases to optimizing compilers. To cite a few of its applications in the domain of programming languages and compilation: redundant synchronization removal, testing the legality of iteration reordering transformations, computing closed form expressions for induction variables, iteration space slicing and code generation [3, 9, 10], model checking and in particular reachability analysis (see [7] and included references), testing equivalence between codes [1, 2, 11].

Graphs can be represented in different ways. One of possible representations of graphs is based on tuple relations. In this paper, we consider the class of parameterized and affine integer tuple relations whose constraints consist of affine equalities and inequalities. Such relations describe infinite graphs. There are many techniques for computing transitive closures for finite graphs, but to our best knowledge, techniques for computing the transitive closure of a parameterized affine integer tuple relation, that describes infinite graphs, were the subject of the investigation of a few papers only [5–7, 10]. The solution presented by Kelly *et al.* [10] is based on heuristics that guarantee neither calculating the exact result nor its conservative approximation.

This paper presents a method to calculate the transitive closure of a relation R being a union of n integer tuple relations R_i, i=1,2,...,n. The domain and range of an integer tuple relation consists of integer tuples. Our approach is based on

computing relation A[R^k], (symbolically) for all k, being either the exact R^k or its overapproaximation. We assume that power k of each individual relation R_i can be computed.

$R^k = \underbrace{R \circ R \circ ... \circ R}_{k\ times}$ represents the power k of relation R.

Computation of R^k itself is an important result. It opens the door for extracting fine-grained parallelism available in program loops, for example, by building the free-scheduling of loop statement instances [8] implying that each loop statement instance is executed as soon as its operands are ready. When R^k does not describe redundant synchronization [10], then to get the free-scheduling we form the set range(R^k), use the constraints of R^k to calculate the upper bound of k, k_{max}, defining the latency of the free-scheduling [8], and generate a nested loop whose outermost nest enumerates values of k from 0 to k_{max} while inner loops scan statement instances to be executed at time k (elements of the set range(R^k)).

Having represented the constraints of R^k as a Presburger formula, we can easily get a representation of the positive transitive closure of R, R^+, by making k in the formula for A[R^k] existentially quantified, i.e., by adding the quantifier "$\exists$" to k into the constraints of R^k.

Having found the positive transitive closure of relation R, R^+, the transitive closure of R, R^*, is calculated as follows

$R^* = R^+ \cup I$, where I is the identity relation.

We propose a necessary and sufficient condition defining when positive transitive closure calculated according to the suggested technique corresponds to exact positive transitive closure. Further on in this paper transitive closure refers to positive transitive closure.

2 Background and Related Work

This section briefly presents some definitions necessary for the rest of the paper and describes related work for transitive closure computation.

An integer k-tuple is a point in Z^k. An integer tuple relation has the following general form {[*input_list*] $\rightarrow$[*output_list*]: *constraints*}, where *input_list* and *output_list* are the lists of variables and/or expressions used to describe input and output tuples, and *constraints* is a Presburger formula describing the constraints imposed upon *input_list* and *output_list* [10].

The transitive closure of a directed graph G=(V, E) is a graph H=(V, F) with edge (v, w) in F if and only if there is a path from v to w in G. There is a path in G if and only if there is an edge (v, w) in H. A graph can be represented with an integer tuple relation whose domain consists of integer k-tuples and whose range consists of integer k′-tuples, for some fixed k and k′.

We use the following standard operations on relations and sets: union ($\cup$), composition ($\circ$ - for a pair of relations, $\prod$ - for multiple relations), inclusion ($\subseteq$), domain(R) (S=domain(R), x$\in S$ iff $\exists$y s.t. {x$\rightarrow$y}$\in R$), range(R) (S=range(R), y$\in$S iff $\exists$x s.t. {x$\rightarrow$y}$\in$R).

Transitive closures of relations with arbitrary Presburger constraints are not computable in general [5, 10]. Two approaches have been studied: finding a particular class of relations for which transitive closure can be computed exactly, and finding an approximation of transitive closure that is "good enough" for a particular class of relations. The general form of a relation is as follows:

$$\left\{ [x] \rightarrow [y] \mid \bigvee_{i=1}^{n} (\exists \alpha_i, A_i(x, y, \alpha_i) \geq b_i) \right\},$$

where x, y, and α_i are integer tuples, A_i is an integer matrix and b_i is a tuple of integer and symbolic values. A distinction is then made when relations contain a single clause (meaning $n = 1$) and are of the form:

$$\{ [x] \rightarrow [y] \mid \exists \alpha, A(x, y, \alpha) \geq b \}$$

and when they contain multiple clauses ($n > 1$).

For single clause relations, the exact formulation of transitive closure has been given in particular cases. For relations of the form $R = \{[x] \rightarrow [Ax+b] \mid Cx \geq d\}$ where A and C are integer matrices, b and d are constant integer vectors, Boigelot in [5] (theorem 8.53, p.236) defines a sufficient condition for the computation of R^*. In briefly, this condition states that if there exists p such that A^p is diagonalizable with eigenvalues in $\{0, 1\}$ then R^* is computable in Presburger arithmetics. In particular, idempotent matrices correspond to transitive closures that are represented by periodic sets [6]. Kelly *et al.* [10] also define a simple class of relations, called d-forms, for which the transitive closure is easily computable. These d-form relations are single clause relations with constraints only on the difference of output and input tuples. Bielecki *et al.* [4] show how to compute the transitive closure of a single relation representing graphs of the chain topology. The exact transitive closure calculation is based on resolving a system of recurrence equations being formed from the input and output tuples of a dependence relation.

For multiple clause relations (or union of single clause relations), the computation of transitive closure is more complex. To our knowledge, it is not yet known, for instance, whether the transitive closure of a union of d-forms is computable. Approaches proposed by Boigelot in [5] or Kelly *et al.* [10] compute approximations of transitive closures for such relations. The union of polyhedra defined by multiple clauses is then approximated into the convex hull of the polyhedra. The multiple clauses are therefore approximated into a single clause. Kelly *et al.* propose a flexible approach based on very specific cases with recipes to compute the transitive closure of a union of relations. The conditions for which the computation is exact are not decided beforehand.

In the following sections, we present a new technique to compute the transitive closure for a union of single clause relations, assuming the power k of each individual relation can be computed.

3 Computing the Power k of a Union of Multiple Relations

In this section, we present an approach to calculate the power k of relation R being a union of single clause relations, denoted as A$[R^k]$. Given a set of $n \geq 2$ relations R_i, i=1,2,...,n, relations $\bar{R}_i$ are obtained by transforming the constraints of R_i so that their domains (and, respectively, ranges) are infinite. When relations $\bar{R}_i$ are commutative (this means that $\bar{R}_i \circ \bar{R}_j = \bar{R}_i \circ \bar{R}_i$ for all i, j), our approach to compute $R^k = (\bigcup_{i=1}^{n} R_i)^k$ is the following

$$A[R^k] = \{[x] \rightarrow [y] \mid x \in \text{domain}(R) \wedge y \in \text{range}(R) \wedge \exists k_i \geq 0, i = 1, 2, ..., n,\ s.t.\ y \in \prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \wedge \sum_{i=1}^{n} k_i = k \wedge k > 0\}. \quad (1)$$

We specify the following classes of relations R_i, i=1,2,...,n, $n \geq 2$, for which the corresponding relations $\bar{R}_i$ are always commutative.

1. Uniform relations, that is, relations of the form $R = \{[x] \rightarrow [x+b] \mid\ Cx \geq d\}$, where C is the integer matrix, b and d are the constant integer vectors;
2. Non-uniform relations of the form $R = \{[x] \rightarrow [Ax] \mid\ Cx \geq d\}$, where A is the diagonal integer matrix, C is the integer matrix, d is the constant integer vector.
3. Relations of the form $R = \{[x] \rightarrow [OP(A,\ x,\ op)] \mid\ Cx \geq d\}$, where A is the diagonal integer matrix, C is the integer matrix, d is the constant integer vector, and OP is the operator producing the following output:

$$OP(A,\ x,\ op) = \begin{vmatrix} a_{11}\ op_1\ x_1 \\ a_{22}\ op_2\ x_2 \\ \quad ... \\ a_{nn}\ op_n\ x_n \end{vmatrix},$$

where

$$A = \begin{vmatrix} a_{11} & 0 & ... & 0 \\ 0 & a_{22} & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & a_{nn} \end{vmatrix},\ x = \begin{vmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{vmatrix},\ op = \begin{vmatrix} op_1 \\ op_2 \\ ... \\ op_n \end{vmatrix},\ op_i \in \{+, \times\},\ 1 \leq i \leq n,$$

op is the same for all relations R_i.

The two previous classes of relations are the special cases of the third class – when op is composed of only the addition operators, we deal with the first class, and when op is composed of only the multiplication operators, we deal with the second class.

To proove that relations $\bar{R}_1 = \{[x] \rightarrow [OP(A,\ x,\ op)]\}$ and $\bar{R}_2 = \{[x] \rightarrow [OP(B,\ x,\ op)]\}$ are indeed commutative for given A, B, x, and op, we compute

$$\bar{R}_1 \circ \bar{R}_2 = \left\{ \left| \begin{array}{c} x_1 \\ x_2 \\ ... \\ x_n \end{array} \right| \rightarrow \left| \begin{array}{c} a_{11}\ op_1\ b_{11}\ op_1\ x_1 \\ a_{22}\ op_2\ b_{22}\ op_2\ x_2 \\ ... \\ a_{nn}\ op_n\ b_{22}\ op_n\ x_n \end{array} \right| \right\},$$

$$\bar{R}_2 \circ \bar{R}_1 = \left\{ \left| \begin{array}{c} x_1 \\ x_2 \\ ... \\ x_n \end{array} \right| \rightarrow \left| \begin{array}{c} b_{11}\ op_1\ a_{11}\ op_1\ x_1 \\ b_{22}\ op_2\ a_{22}\ op_2\ x_2 \\ ... \\ b_{nn}\ op_n\ a_{22}\ op_n\ x_n \end{array} \right| \right\}.$$

Because the addition/multiplication operators represented by each op_i, $1 \leq i \leq n$, are commutative and associative, we can conclude that $\bar{R}_1 \circ \bar{R}_2 = \bar{R}_2 \circ \bar{R}_1$, that is, relations $\bar{R}_1$ and $\bar{R}_2$ are commutative.

For example, relations $\bar{R}_1$=$\{[i,j] \rightarrow [i+1, 2*j]\}$ and $\bar{R}_2$=$\{[i,j] \rightarrow [i+3, 5*j]\}$ are commutative (they belong to the third class with $op = [+\ *]$), while relations $\bar{R}_3$=$\{[i,j] \rightarrow [2*i, j+1]\}$ and $\bar{R}_4$=$\{[i,j] \rightarrow [i+3, 5*j]\}$ are not commutative.

For relations of the general form $R = \{[x] \rightarrow [Ax + b] \mid Cx \geq d\}$, the satisfaction of the commutativity condition introduced above depends on particular values of A and b and should be verified for each pair of relations R_i and R_j by checking whether $\bar{R}_i \circ \bar{R}_j$ is equal to $\bar{R}_i \circ \bar{R}_i$ using any tool permitting for operations on relations.

The following property guarantees that A[R^k] defined by (1) is either the exact representation of R^k or its overapproximation.

Property 1 $R^k \subseteq A[R^k]$.

Proof. By definition, $R^k = \prod_{j=1}^{k} \bigcup_{i=1}^{n} R_i$. Since $R_1 \subseteq \bar{R}_1$, $R_2 \subseteq \bar{R}_2$, ... $R_n \subseteq \bar{R}_n$, the following is true:

$$\prod_{j=1}^{k} \bigcup_{i=1}^{n} R_i \subseteq \prod_{j=1}^{k} \bigcup_{i=1}^{n} \bar{R}_i.$$

Because relations $\bar{R}_1$, $\bar{R}_2$, ..., $\bar{R}_n$ commute, we can group all the occurrences of the same $\bar{R}_i$ together. It can be shown using properties of a commutative semiring on relations that

$$\prod_{j=1}^{k} \bigcup_{i=1}^{n} \bar{R}_i = \bigcup_{\substack{k_1, k_2, ..., k_n \\ \sum k_i = k, k_i \geq 0}} \bar{R}_1^{k_1} \circ \bar{R}_2^{k_2} \circ ... \circ \bar{R}_n^{k_n}.$$

It implies that $\forall$ x,y s.t. {x$\rightarrow$y}$\in R^k$,

$$\exists k_i \geq 0, i = 1, 2, ..., n,\ s.t.\ y \in \prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \wedge \sum_{i=1}^{n} k_i = k \wedge k > 0.$$

Because {x→y}∈ R^k means that x∈ domain(R) and y∈ range(R), we can conclude that $R^k \subseteq \mathrm{A}[R^k]$.

Note that for proving the property $R^k \subseteq \mathrm{A}[R^k]$ we require that relations $\bar{R}_1$, $\bar{R}_2$, ..., $\bar{R}_n$ be commutative. Under this condition, for $\mathrm{A}[R^+]$ formed by making k in the constraints of $\mathrm{A}[R^k]$ existentially quantified, we can conclude that $R^+ \subseteq \mathrm{A}[R^+]$ meaning that $\mathrm{A}[R^+]$ is either the exact representation of R^+ or its overapproximation.

The following theorem provides a necessary and sufficient condition when $\mathrm{A}[R^k] = R^k$.

Theorem 1 *$A[R^k]$=R^k if and only if the following condition is satisfied for all positive values of k:*

$$\forall x \in \mathit{domain}(R) \cup \mathit{range}(R) \ \wedge \ \forall k_i \geq 0, i = 1, 2, ..., n, \ \mathit{s.t.} \ \sum_i k_i = k \ \wedge \ k > 0,$$
$$(\prod_{i=1}^{n} \bar{R}_i^{k_i}(x) \in \mathit{range}(R)) \Rightarrow (\exists h_i \geq 0, i = 1, 2, ..., n, \ \mathit{s.t.} \ \sum_{i=1}^{n} h_i = k, \qquad (2)$$
$$\prod_{i=1}^{n} \bar{R}_i^{k_i}(x) = \prod_{i=1}^{n} \bar{R}_i^{h_i}(x) \ \wedge \ x \in \bigcup_{i=1, h_i>0}^{n} \mathit{domain}(R_i)).$$

Proof. The proof consists of two steps. We first prove by induction that Condition (2) is the sufficient condition for $\mathrm{A}[R^k]$ to be the exact representation of R^k. Next, we prove that Condition (2) is the necessary condition.

Sufficient condition: (2) $\Rightarrow$ $\mathbf{A}[R^k] \subseteq R^k$.

Consider x,y s.t. {x→y}∈$\mathrm{A}[R^1]$. The satisfaction of Condition (2) guarantees that $\exists i$ s.t. y∈ $\bar{R}_i$(x) and x∈ domain(R_i). This means that {x→y}∈ R^1, i.e., $\mathrm{A}[\mathrm{R}^1] \subseteq R^1$.

Now, assuming that $\mathrm{A}[R^k] = R^k$, we want to prove that $\mathrm{A}[R^{k+1}]$=R^{k+1}.
Consider x,y s.t. {x→y}∈$\mathrm{A}[R^{k+1}]$. By definition of $\mathrm{A}[R^{k+1}]$,
x∈domain(R), y∈range(R), $\exists k_i \geq 0$, i=1,2,...,n,
s.t. $\sum_{i=1}^{n} k_i = k + 1$, y∈ $\prod_{i=1}^{n} \bar{R}_i^{k_i}(x)$.
The satisfaction of Condition (2) guarantees that
$\exists h_i \geq 0$, i=1,2,...,n, s.t. $\sum_{i=1}^{n} h_i = k+1$,
y∈ $\prod_{i=1}^{n} \bar{R}_i^{h_i}(x) \in \mathrm{range}(R)$, x∈ $\bigcup_{i=1, h_i>0}^{n} \mathrm{domain}(R_i)$.
In particular, $\exists j \in [1,2,...,n]$ such that $h_j > 0$, x∈domain(R_j).

Since relation R_j commutes with the remaining relations, we have y$\in \prod_{i,i\neq j} \bar{R}_i^{h_i} \circ \bar{R}_j^{h_j}$(x).

Consider z$\in \bar{R}_j$(x) such that y$\in \prod_{i,i\neq j} \bar{R}_i^{h_i} \circ \bar{R}_j^{h_j-1}$(z). Because x$\in$ domain(R_j) and z$\in$ range(R_j), the following is true: {x$\rightarrow$z}$\in$R. Moreover, because Condition (2) is satisfied for z, {z$\rightarrow$y}$\in$ A[R^k]. By induction, {z$\rightarrow$y}$\in$ R^k. Thus, {x$\rightarrow$y}$\in$ R^{k+1} and A[R^{k+1}]$\subseteq$ R^{k+1}.

Necessary condition: A$[R^k] \subseteq R^k \Rightarrow$ (2).

Let us now show that Condition (2) of Theorem 1 is the necessary condition for A[R^k] to be the exact representation of R^k. Assume Condition (2) is not satisfied but A[R^k] $\subseteq$ R^k.

Since A[R^k] $\subseteq$ R^k, {x$\rightarrow$y}$\in$ R^k. This implies that x$\in$ domain(R), y$\in$ range(R) and $\exists i_h \in$[1,2,...,n], h=1,2,...,k, such that y$\in$ $R_{i_1} \circ R_{i_2} \circ ... \circ R_{i_k}$(x). Note that x$\in$ domain(R_{i_k}). Since $R_1 \subseteq \bar{R}_1$, $R_2 \subseteq \bar{R}_2$, ..., $R_n \subseteq \bar{R}_n$, the following is true: y$\in \bar{R}_{i_1} \circ \bar{R}_{i_2} \circ ... \circ \bar{R}_{i_k}$(x).

Because all relations R_i commute, we conclude that

$\exists$ h$_i \geq$0, i=1,2,...,n, s.t. $\sum_{i=1}^{n} h_i = k$, y$\in \prod_{i=1}^{n} \bar{R}_i^{h_i}$(x).

Such a redefinition of y as well as the knowledge that x$\in$ domain(R_{i_k}) is in contradiction with the assumption that Condition (2) is not satisfied. This proves that indeed Condition (2) is the necessary condition for A[R^k] to be the exact representation of R^k.

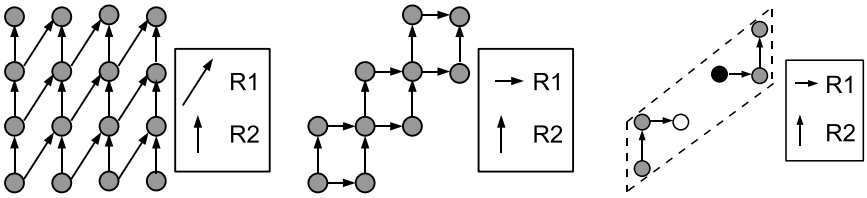

Fig. 1. Examples of graphs (i)

Figure 1 shows three examples of graphs. Condition 2 is satisfied for the first and second graphs, while for the third graph it is not valid: there exists the point x=(2,2) (it is shown in white) for which the condition does not hold. Indeed, for the pair x=(2,2) and y=(3,3) (it is shown in black), we can see that y=$\bar{R}_1 \circ \bar{R}_2$(x)=$\bar{R}_2 \circ \bar{R}_1$(x), but x$\notin$ domain(R_1) and x$\notin$ domain(R_2).

Figure 2 presents two more examples of graphs. Condition (2) is satisfied for the left-hand side graph, while for the right-hand side graph the condition

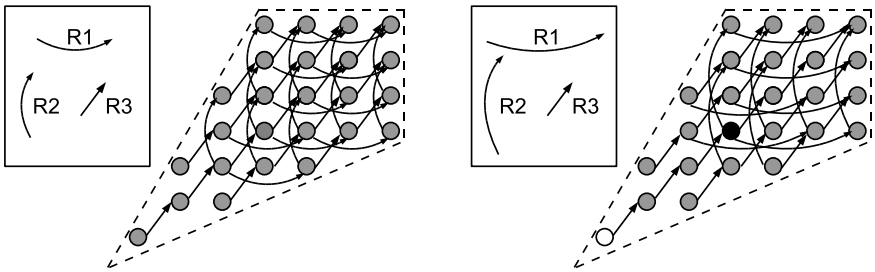

Fig. 2. Examples of graphs (ii)

does not hold: there exists the point x=(1,1) (it is shown in white) for which the condition is not satisfied. Indeed, for the pair x=(1,1) and y=(4,4) (it is shown in black), we have {x→y}∈A[R^2] (y=$\bar{R}_1 \circ \bar{R}_2$(x)=$\bar{R}_2 \circ \bar{R}_1$(x)), but {x→y}$\notin R^2$ (x$\notin$ domain(R_1) and x$\notin$ domain(R_2)). In the matter of fact, {x→y}$\in R^3$, because y=$R_3 \circ R_3 \circ R_3$(x).

4 Illustrating Examples

In this section, we consider several examples illustrating the presented approach and compare our results with those yielded by approaches presented in paper [10]. We use the Omega syntax [12] to present relations and sets in our examples and the Omega calculator to carry out necessary calculations. Having computed A[R^k], we want to check whether Condition (2) is satisfied. For this purpose, we calculate set X containing elements that satisfy the left-hand side of the implication of Condition (2) and do not satisfy the right-hand side of this implication. We build relation $X1$ whose domain containts elements satisfying the left-hand side of the implication. Next, we build relation $X2$ whose domain contains elements not satisfying the right-hand side of the implication. The computation of the domain($X1 - X2$) defines set X. If set X is empty, then we conclude that A[R^k] corresponds to the exact representation of R^k.

Steps to check whether Condition (2) is valid are the following.

1. Compute A[R^k] and A[R^+].
2. Check whether A[R^k] = R^k. In the matter of fact,
   (a) form the following relation $X1$

$$X1 = \{x \to y \mid x \in \text{domain}(R) \cup \text{range}(R),\ y \in \text{range}(R),\ \exists k_i \geq 0\ s.t.\ \sum_{i=1}^{n} k_i = k,\ y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)\};$$

   (b) form the following relation $X2$

$$X2 = \{x \to y \ \mid\ x \in \text{domain}(R) \cup \text{range}(R),\ y \in \text{range}(R),\ \exists k_i \geq 0\ \ s.t.\ \sum_{i=1}^{n} k_i = k,\ y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x),\ x \in \bigcup_{i=1,k_i>0}^{n} \text{domain}(R_i)\};$$

(c) calculate the set $X = \text{domain}(X1 - X2)$. If set X is empty, $\text{A}[R^k] = R^k$ and, respectively, $\text{A}[R^+]=R^+$. The end. Otherwise, $\text{A}[R^k] \neq R^k$.

To illustrate the proposed approach, we start with the following example.

Example 1.

R_1:={[i,j]→[i+1,j+1]: 1≤i<n & 1≤j<n},
R_2:={[i,j]→[i+1,j-1]: 1≤i<n & 2≤j≤n}.

Figure 3 illustrates the graph described with relations R_1 and R_2 for n=5. Our task is to compute relation $\text{A}[R^k]$ and to find $\text{A}[R^+]$ using $\text{A}[R^k]$. Relations $\bar{R}_1$ and $\bar{R}_2$ derived from relations R_1 and R_2 by transforming their constraints so that their domains (and, respectively, ranges) are infinite have the following form

$\bar{R}_1$:={[i,j]→[i+1,j+1]},
$\bar{R}_2$:={[i,j]→[i+1,j-1]}.

$\bar{R}_1$ and $\bar{R}_2$ are commutative because $\bar{R}_1 \circ \bar{R}_2 - \bar{R}_2 \circ \bar{R}_1 = \emptyset$ and $\bar{R}_2 \circ \bar{R}_1 - \bar{R}_1 \circ \bar{R}_2 = \emptyset$ (this is easy to check by means of the Omega calculator).

$\bar{R}_1^{k_1}$, $\bar{R}_2^{k_2}$ can be calculated easily using the formula presented in [10] and they are given below.

$\bar{R}_1^{k_1}$:={[i,j]→[i′,j′]: i′=i+k_1 & j′=j+k_1 & k_1 ≥0 },
$\bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i+k_2 & j′=j-k_2 & k_2 ≥0 }.

The composition $\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$ is the following

$\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i+$k_1$+$k_2$ & j′=j+k_1-k_2 & k_1 ≥0 & k_2 ≥0 }.

Finally, we are able to calculate $\text{A}[R^k]$ that can be represented as

$\text{A}[R^k]$:={[i,j]→[i′,j′]: 1≤i<n & 1≤j≤n & 2≤i′ ≤n & 1≤j′ ≤n & Exists (k_1,k_2: i′=i+k_1+k_2 & j′=j+k_1-k_2 & k_1 ≥0 & k_2 ≥0 & $k_1 + k_2 = k$ & k ≥0)}.

In order to check whether Condition (2) is satisfied, we form relations $X1$ and $X2$, and calculate set X.

$X1$:={[i,j]→[i′,j′]:

$$\underbrace{1 \leq \text{i}, \text{j} \leq n}_{x \in \text{domain}(R) \cup \text{range}(R)} \quad \& \quad \underbrace{2 \leq \text{i}' \leq n \;\&\; 1 \leq \text{j}' \leq n}_{y \in \text{range}(R)} \;\&$$

$$\text{Exists}(k_1, k_2 : \underbrace{\text{i}' = \text{i} + k_1 + k_2 \;\&\; \text{j}' = \text{j} + k_1 - k_2}_{y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)} \;\&$$

$$k_1 \geq 0 \;\&\; k_2 \geq 0 \;\&\; k_1 + k_2 = k \;\&\; k > 0)\};$$

$X2$:={[i,j]→[i′,j′]:

$$\underbrace{1 \leq \text{i}, \text{j} \leq n}_{x \in \text{domain}(R) \cup \text{range}(R)} \quad \& \quad \underbrace{2 \leq \text{i}' \leq n \;\&\; 1 \leq \text{j}' \leq n}_{y \in \text{range}(R)} \;\&$$

$$\text{Exists}(k_1, k_2 : \underbrace{\text{i}' = \text{i} + k_1 + k_2 \;\&\; \text{j}' = \text{j} + k_1 - k_2}_{y = \prod_{i=1}^{n} \bar{R}_i^{k_i}(x)} \;\&$$

$$k_1 \geq 0 \;\&\; k_2 \geq 0 \;\&\; k_1 + k_2 = k \;\&\; k > 0 \;\&$$

$$(\underbrace{1 \leq \mathrm{i,j} < n \;\&\; k_1 > 0}_{x \in \mathrm{domain}(R_1),\ k_1 > 0} \text{ OR } \underbrace{1 \leq \mathrm{i} < n \;\&\; 2 \leq \mathrm{j} < n \;\&\; k_2 > 0}_{x \in \mathrm{domain}(R_2),\ k_2 > 0}))\};$$

$X := \mathrm{domain}(X1 - X2) = \emptyset$.

Because set X is empty, Condition (2) is satisfied and $\mathrm{A}[R^k] = R^k$ meaning that $\mathrm{A}[R^k]$ found by means of our approach is the exact representation of R^k.

To get the positive transitive closure, R^+, of the union of R_1 and R_2, we make k in the formula for $\mathrm{A}[R^k]$ to be existentially quantified, i.e.,

R^+:={[i,j]→[i′,j′]: 1≤i<n & 1≤j≤n & 2≤i′ ≤n & 1≤j′ ≤n & Exists (k_1, k_2, k: i′=i+$k_1 + k_2$ & j′=j+$k_1 - k_2$ & $k_1 \geq 0$ & $k_2 \geq 0$ & $k_1 + k_2 = k$ & $k \geq 0$)}.

Applying the Omega calculator implementing the approach to compute transitive closure presented in paper [10], we yield a complex representation of transitive closure including the seven "union" operators, while the both forms (ours and produced by Omega) represent the exact transitive closure.

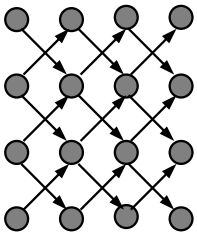

Fig. 3. Graph of Ex.1

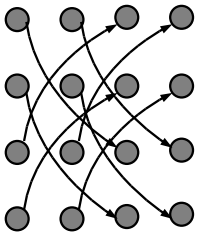

Fig. 4. Graph of Ex.2

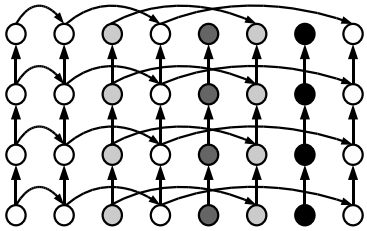

Fig. 5. Graph of Ex. 3

Let us now consider the following set of relations being a slight modification of the previous example. We will show that Omega is fragile to even the simplest modification of Example 1.

Example 2.

R_1:={[i,j]→[i+2,j+2]: 1≤i<n-1 & 1≤j<n-1},

R_2:={[i,j]→[i+2,j-2]: 1≤i<n-1 & 3≤j≤n}.

Figure 4 illustrates the graph described with relations R_1 and R_2 for n=5. Using the Omega calculator, it is easy to state that the corresponding relations $\bar{R}_1$ and $\bar{R}_2$ are commutative and Condition (2) is satisfied.

Applying our approach, we get

R^+:={[i,j]→[i′,j′]: 1≤i<n-1 & 1≤j≤n & 3≤i′ ≤n & 1≤j′ ≤n & Exists (k_1,k_2,k: i′=i+$2 * k_1 + 2 * k_2$ & j′=j+$2 * k_1 - 2 * k_2$ & $k_1 \geq 0$ & $k_2 \geq 0$ & $k_1 + k_2 = k$ & $k \geq 0$)},

while applying the approach implemented in the Omega calculator [10] it is not possible to obtain any result.

Let us now show how the proposed method can be applied to non-uniform relations.

Example 3.
R_1:={[i,j]→[2i,j]: 2≤2i≤n & 1≤j≤m},
R_2:={[i,j]→[i,j+1]: 1≤i≤n & 1≤j<m}.

Figure 5 illustrates the graph described with relations R_1 and R_2 for n=8 and m=4. Relations $\bar{R}_1$ and $\bar{R}_2$ derived from relations R_1 and R_2 by transforming their constraints so that their domains (and, respectively, ranges) are infinite have the following form
$\bar{R}_1$:={[i,j]→[2i,j]},
$\bar{R}_2$:={[i,j]→[i,j+1]}.

$\bar{R}_1$ and $\bar{R}_2$ are commutative because $\bar{R}_1 \circ \bar{R}_2 - \bar{R}_2 \circ \bar{R}_1 = \emptyset$ and $\bar{R}_2 \circ \bar{R}_1 - \bar{R}_1 \circ \bar{R}_2 = \emptyset$.

Because relation R_1 is non-uniform, we cannot compute $\bar{R}_1^{k_1}$ using Omega. But using Mathematica according to the approach described in [4] we yield:

$\bar{R}_1^{k_1}$:={[i,j]→[i′,j′]: (i′=i^{2*k_1} & $k_1 \geq 1$ OR i′=i & k_1=0) & j′=j },

while $\bar{R}_2^{k_2}$ (computed either using Omega or according to [4]) is the following:

$\bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: i′=i & j′=j+k_2 & $k_2 \geq 0$ }.

The composition $\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$ is of the form
$\bar{R}_1^{k_1} \circ \bar{R}_2^{k_2}$:={[i,j]→[i′,j′]: (i′=i^{2*k_1} & $k_1 \geq 1$ OR i′=i & k_1=0) & j′=j+k_2 & $k_2 \geq 0$}.

Finally, we are able to calculate A[R^k] that can be represented as
A[R^k]:={[i,j]→[i′,j′]: (1≤i≤n & 1≤j<m OR j=m & 1≤i & 2*i≤n) &
(1≤i′ ≤n & 2≤j′ ≤m OR Exists (alpha: 2*alpha = i′ & 2≤i′ ≤n & j′=1)) &
Exists (k_1,k_2: (i′=i^{2*k_1} & $k_1 \geq 1$ OR i′=i & k_1=0) & j′=j+k_2 & $k_2 \geq 0$ & k_1+k_2 = k & $k \geq 0$)}.

Using Mathematica, it can be shown that set X is empty for this example. Thus, A[R^k]=R^k and A[R^+] found making k in the formula for A[R^k] to be existentially quantified is the exact representation of R^+, i.e., A[R^+]=R^+.

5 Conclusion and Future Work

We presented in this paper a novel approach for the computation of the transitive closure of a union of affine and parameterized relations. For relations that commute when their domains and ranges are set to be infinite, we proposed a formulation for both the power k and the positive transitive closure of the union of $n \geq 2$ relations, for a symbolic k. The precise class of relations for which the computation is exact is defined by the necessary and sufficient condition presented in Section 3. This class includes in particular non-convex relations. For relations beyond this class, we proved that the approach still provides an overapproximation of transitive closure.

In comparison to the heuristic approach presented in [10], ours always permits the computation of the transitive closure of a union of multiple relations. A result can be either exact transitive closure or its overapproximation, depending on the satisfaction of the introduced condition.

In our future research we plan to derive approaches for calculating transitive closure for a union of $n \geq 2$ relations R_i, i=1,2,...,n, whose corresponding relations $\bar{R}_i$ are **not** commutative.

References

1. Christophe Alias and Denis Barthou. On Domain Specific Languages Re-Engineering. In *ACM Int. Conf. on Generative Programming and Component Engineering*, volume 3676 of *Lect. Notes in Computer Science*, pages 63–77, Tallinn, September 2005. Springer-Verlag.
2. Denis Barthou, Paul Feautrier, and Xavier Redon. On the equivalence of two systems of affine recurrence equations. In *Euro-Par Conference*, volume 2400 of *Lect. Notes in Computer Science*, pages 309–313, Paderborn, August 2002. Springer-Verlag.
3. Anna Beletska, Wlodzimierz Bielecki, and Pierluigi San Pietro. Extracting coarse-grained parallelism in program loops with the slicing framework. In *ISPDC '07: Proceedings of the Sixth International Symposium on Parallel and Distributed Computing*, page 29, Washington, DC, USA, 2007. IEEE Computer Society.
4. Wlodzimierz Bielecki, Tomasz Klimek, and Konrad Trifonuwic. Calculating exact transitive closure for a normalized affine integer tuple relation. *To be published in the Journal of Electronic Notes in Discrete Mathematics*, 2009.
5. B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1998.
6. B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proceedings of the 6th International Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer-Verlag, 1994.
7. Hubert Comon and Yan Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV'98, LNCS 1427*, pages 268–279. Springer, 1998.
8. Alain Darte, Yves Robert, and Frederic Vivien. *Scheduling and Automatic Parallelization.* Birkhaüser, 2000.
9. Wayne Kelly, Vadim Maslov, William Pugh, Evan Rosser, Tatiana Shpeisman, and David Wonnacott. The omega library interface guide. Technical report, College Park, MD, USA, 1995.
10. Wayne Kelly, William Pugh, Evan Rosser, and Tatiana Shpeisman. Transitive closure of infinite graphs and its applications. *Int. J. Parallel Programming*, 24(6):579–598, 1996.
11. K. C. Shashidhar, Maurice Bruynooghe, Francky Catthoor, and Gerda Janssens. An automatic verification technique for loop and data reuse transformations based on geometric modeling of programs. *Journal of Universal Computer Science*, 9(3):248–269, March 2003.
12. The Omega project. `http://www.cs.umd.edu/projects/omega`.